Mid Sweden University
The Department of Information Technology and Media (ITM)
Author: David Rutberg
E-mail address: david.rutberg@gmail.com
Study programme: Bachelor of Science, 180 hp
Examiner: Ulf Jennehag, Ulf.Jennehag@miun.se
Tutor: Rahim Rahmani, Rahim.Rahmani@miun.se
Tutor: Mats Nordlund, Absilion AB, Mats.Nordlund@absilion.com
Scope: 8933 words inclusive of appendices
Date: 8/21/10

B.Sc. Thesis
within Computer Engineering C, 15 higher education credits

Aggregation and visualization of test data

David Rutberg

Abstract

New Internet services place new demands on computer networks. To test that the networks can handle these new services the company Absilion has developed a test system called One Button Tester. To make the test system even easier to use an aggregation functionality is required. The main function of this thesis work is to show how to design and implement such functionality into the test system. The requirement is that a non-advanced network technician could determine, both rapidly and easily, whether the test is a failure or a success. Bearing in mind modern theories about programming a prototype has been developed that is an addition to the existing system. The prototype manages test data retrieval, aggregation calculation of test data and storage of aggregated values in a database. This is all conducted with its priorities being both performance and re-usability. Finally a visual representation of the aggregated values was constructed that makes it easy to rapidly determine the result of the test.

Table of Contents

1 Introduction

1.1 Background and problem motivation

As the Internet services advance so will new demands on the computer networks. For example, services such as Internet telephony and Internet protocol television can already be seen to be making their way into the computer networks.

Compared to ordinary web browsing these services have some additional characteristics that the computer networks must support [1][2].

It is possible to overlook certain network issues when waiting for a web page to load even if it proves to be an annoyance. However, the same issue for Internet telephony may cause the whole conversation to disconnect entirely with the result that the conversation must be re-established. This would not be considered to be the expected behaviour for a user who has switched from the old telephone network to making their phone calls over the computer networks (Internet).

This would be a similar scenario in relation to Internet Television. Imagine watching the world championship final penalty shots in soccer when network issues makes the picture freeze and then skips ahead a few seconds. Scenarios such as this might cause a customer with a particular Internet Service Provider (ISP) to instantly switch to another ISP.

Techniques do actually exist for building more reliable networks to handle these new services [3]. The question is how to test for it?

In order for the ISPs to effectively test their networks for Internet telephony and Internet protocol television before it is launched to customers the company Absilion has developed a network tool called OBT (One Button Tester) [4]. The OBT is a distributed web-based system and the intention is that it should be very easy to install and use. The test data that is collected by the OBT is processed and displayed on an online web page [4].

As Absilion AB attempts to make it even easier for a user of the OBT system to recognize problems in the networks, it was necessary to discover a means of aggregating and visualizing a set of test data.

The challenge is how to build an addition to the existing OBT system that can deal with the aggregation of test data and then visualize the result of the aggregation. The visualization should be performed in such a way that makes it really easy to determine whether or not the network performs in a satisfactory manner.

1.2 Overall aim

The main goal for this thesis project is to make it easier for an non-advanced network technician to examine the test results for Internet telephony and Internet protocol television. The technician should be able to discover errors in the network both live as the test runs and after the test is finished. Only a quick glance at a computer screen should be necessary in order to determine the results of the test. In the case of network errors the technician should be able to navigate in the GUI to discover the source test data that has caused the test to fail.

Firstly, an understanding of the current system is required. This is required in order to develop an addition to the existing system that handles both the retrieval and aggregation of the test data.

Secondly an investigation regarding how to make an easy to comprehend the visual representation of the aggregated data will be performed.

Finally it is all connected to the existing OBT system in a test environment. The end result should be a web GUI (Graphical User Interface) that shows a visualization of the aggregated data for tests of both Internet telephony and Internet protocol television.

1.3 Scope

The focus of the project is to construct, implement and test a prototype that can aggregate and visualize test data on a web page. The project falls within the software engineering field of computer science. The project is not concerned about the actual network tests and the techniques and theories within the network field of computer science.

1.4 Concrete and verifiable goals

The goals of the study are to,

- Obtain an understanding of the existing OBT system architecture and usage.
- Collect test data from the OBT system to use in the aggregation.
- Modify existing test-scripts to support aggregation functionality.
- Find a way to calculate aggregation of test data depending on the type of test being performed.
- Make it possible to store aggregation results in a database.
- Retrieve aggregated data from a database and make it available in the web page dynamically.
- Construct an easy to understand visualization of the aggregation result.
- Create levels of aggregation.
- Make a easy to use navigation between levels of aggregation and test data on web page.

1.5 Outline

Chapter 2 is about the methods used during the thesis project. Chapters 3 and 4 deal with the theory of the thesis work and describe the Internet services. These chapters also review how the OBT system is constructed and how to use it. Chapter 5 performs an analysis of the problem and

states the requirements specification. After that the result of the thesis work is presented in chapter 6. Finally a discussion and conclusions of the thesis project is available in chapter 7.

2 Methodology

2.1 Theoretical studies

Theoretical studies were performed on basic computer networking theory and Internet services such as Internet telephony and Internet protocol television. To gain a basic understanding of computer networks and Internet services Wikipedia articles were studied and these were complemented by academic literature on the subject.

For Python programming theory and best practices in web development and object oriented programming the online Django web framework documentation [5] and the Python documentation [6] was used.

2.2 Study of the OBT system

As one of the major parts of this thesis project is to implement new functionality to an existing system a thorough understanding of that system is required. This is conducted by gradually investigating the system architecture and the source code of the system in more depth. The use of pen and paper combined with non-formal Unified Modeling Language (UML) has been a good means of building an understanding of an existing system for the author from earlier projects. The important figures and models are included so that the reader can have a better understanding of the report which has been conducted by the rework and software enhanced versions of the pen and paper productions.

To aid in the study of the OBT, Absilion has made their platform for documentation and source code browsing (Trac) available to the author of the report. By using the online source code browser and the wiki functionality the author has been able to browse the source code and easily find the corresponding documentation of the OBT.

2.3 Test environment

As the development of the thesis project will be performed separately from the production system it will be necessary to utilize a test environment. The test environment could be created by the use of a virtualization software called Virtualbox.

To imitate the production environment one physical host machine and at least three virtual machines are required. The first virtual machine is used as a server in which the OBT server software is installed. The other two virtual machines are clients and are used to mimic the network probes that communicates with each other and generate test data. Test data that will be sent to the web server [4].

To enable the virtual machines to communicate with each other a virtual network is created [7]. The virtual machines and the virtual networking between them produce what will now be called a *virtual world*. This virtual world is shown inside the dotted square in Figure 1.

The server can be connected both to the virtual network within the virtual world but also have a second interface connected to the Internet [7]. In this manner a browser can connect to the server from outside the virtual world. This will be used to connect and test new features on the OBT web page with a web browser as the development progresses. The reason not to use a web browser from a virtual machine in the virtual world is because all virtual machines operating systems are text based.

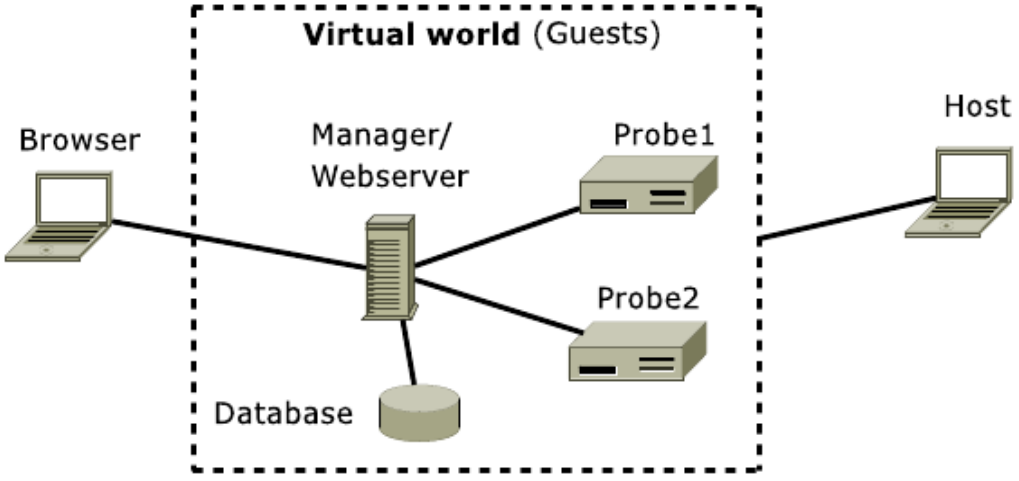

Figure 1: Virtual world

The actual programming will be performed on the host machine. The source code is then copied to the virtual machines through a feature called *shared folders* in Virtualbox [8].

2.4 Software tools and programming languages

2.4.1 Test environment tools

The test environment uses the following software,

- Debian GNU/Linux based distributions for the server and probes virtual machines.
- Archlinux host system running Virtualbox virtualization software.
- Opera, Mozilla Firefox and Konqueror web browsers are used during development and testing.

2.4.2 Python programming language

The OBT system is developed in the Python programming language. Python is a dynamic language which can be used for many different programming tasks including web development and network programming [9]. Python has support for many programming paradigms such as Object Orientated programming, Imperative programming and Functional programming [10]. In this project an Object Oriented approach is used.

2.4.3 Server tools

The server makes use of the MySQL database management system, the Python based web framework Django and the Apache web server. The popular term LAMP (Linux, Apache, MySQL and Python/PHP) could be used to describe the software package as the GNU/Linux operating system is used.

2.4.4 Django web framework

Django is a web framework that makes use of a software development pattern called Model Template View (MTV). The MTV pattern is closely related to the better known Model View Controller (MVC) pattern but with a different approach regarding both the view and controller [11]. The model in Django is the database abstraction layer which makes use of the Django Object Relation Mapper (ORM) to treat relational database tables as ordinary Python objects. The ORM makes it easy to handle database connections and to query the database with Python code requiring no raw Structured Query Language (SQL) baked into the code [12]. The templates are what the user actually views on the screen as it is basically HyperText Markup Language (HTML) processed with the Django template language to display or collect information [13]. In the MVC pattern the Django template would be called the view, however, in MTV the view is the controller of MVC. The view in Django is used to retrieve or write to the database, process information or practically anything required in relation to the Python language [14]. The processed data is then sent from the view to the template system and displayed to the user. The javascript framework Dojo is used to enhance the web GUI user experience with the use of AJAX (Asynchronous JavaScript and XML).

2.4.5 UML and figures

As the Object Oriented paradigm is used the Unified Modeling Language (UML) is used to create figures for classes, objects and relations between them. Umbrello UML editor, Dia and Inkscape software are used to create the figures for the report.

2.4.6 Absilion discussions

A continuing discussion with the management and developers at Absilion is held in order to evaluate and discuss ideas relating to the aggregation functionality implementation and visualization.

In addition the meetings are held to check the progress of the work and to discuss issues emerging during the development.

2.5 Test plan

Since the main part of the thesis project is to actually visualize test data, the result of that visualization will be tested against the requirements for ease of comprehension in terms of errors in the network.

To verify that the aggregation data have been correctly calculated they are compared to the manually computed aggregations of the test data produced by the OBT system.

3 Triple play Internet services

Triple play is a term that is used for a package of services that is distributed on a single broadband[1] connection. Triple play includes Internet access, Internet telephony, and Internet protocol television [2].

3.1 Internet access

For Internet access to be part of triple play a high speed (broadband) Internet connection of at least 256 kb/s [14] is required (but observe that this speed would not suffice for IPTV).

The OBT performs tests with regards to the Internet access best effort throughput [15]. Best effort in this context means that there are no guarantees that a packet reaches its destination in some fixed speed or time (TCP/IP controls that the packet reach it's destination) [16].

3.2 Internet telephony

Internet telephony, or VoIP (Voice over IP), is the second service which forms part of the triple play. By using a computer network instead of the Public Switched Telephone Network (PSTN) as a means of establishing phone calls, new possibilities for telephoning emerge. However this can also bring problems where the main problem is that no QoS (Quality of Service) on ordinary computer networks exist [17].

OBT can test whether an implementation of QoS is working well by the means of the MOS (Mean Opinion Score) [15].

MOS is a calculated value between 1 and 5 where 5 is excellent and 1 is bad [18]. The MOS has to be 4 or above for a test to pass in the OBT system [19].

1 broadband is informally defined to be at least 256 kb/s

3.3 Internet protocol television

The last service in Triple play is the Internet protocol television or IPTV. IPTV is a means of delivering television over IP networks [20]. The networks associated with this should have some characteristics that have the ability to be tested with the OBT,

- Low Delay variation
- Low Loss ratio

As seen above it is delay and loss that have to be tested in a network that supports IPTV.

4 One Button Tester (OBT)

4.1 System overview

When studying the OBT system the system can be broken down into three main parts,

1. Web based GUI (Graphical User Interface)
2. TECS (Test Execution Control Server)
3. Genalyzers (also called probes)

Figure 2 shows an overview of the OBT system.

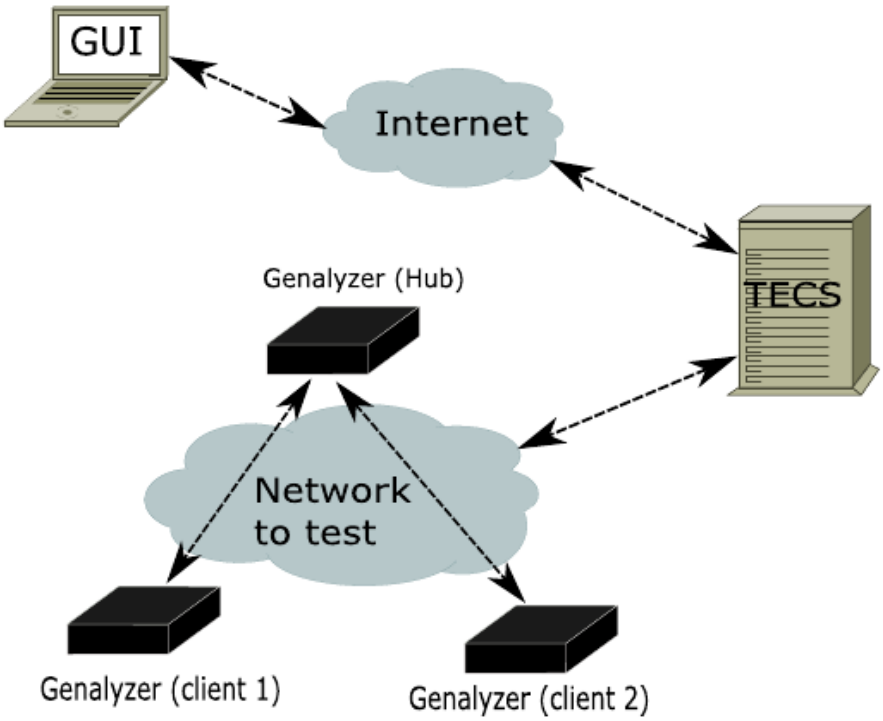

Figure 2: System overview

4.2 Web based GUI

The GUI is where the user interacts with the OBT in order to create tests, control running test results, review old tests and check the status of probes connected to the network [21].

4.2.1 Basic usage

A brief overview regarding the usage of the web GUI is provided below.

- Connect to the OBT homepage through the Internet.
- Create a new job.
- Specify which tests will be executed.
- Specify parameters for the chosen tests.
- Specify which probes to use in the tests.
- Start the tests and follow the results live as they show up in the GUI.
- Evaluate the results of the tests when finished.

4.2.2 The home page

When connecting to the website a simple login page will be presented to the user. The user enters the login information (username and password) supplied by Absilion. After logging in the user will be redirected to *Home*. At *home* the user can obtain a status regarding the jobs that are in progress, recently finished jobs and periodic jobs. The jobs possess a status icon as shown in Figure 4 and where for example *Thesis example4* has a play icon, *Thesis example 3* has a failure icon, *Thesis example* has a passed test icon and the *Thesis example 5* has the scheduler icon.

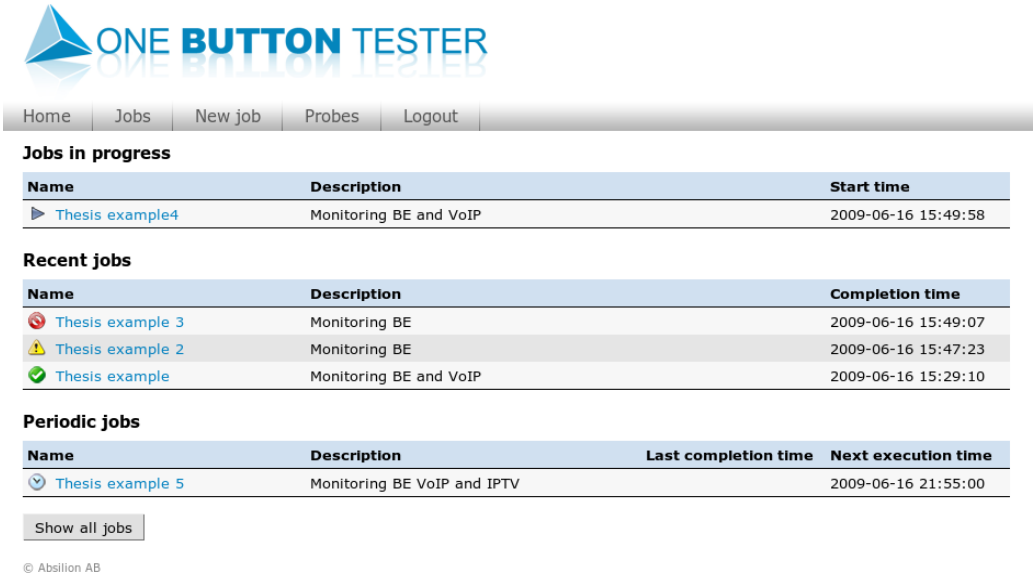

Figure 3: Home in the OBT GUI

4.2.3 The probes page

By clicking the *Probes* button in the GUI menu bar the user can see which probes (genalyzers) are online, in use and which probes are presently offline. This is shown in Figure 4 where probe1 is *in_use* (orange circle in front), probe2 is *free* (green circle) and probe3 is *offline* (red circle).

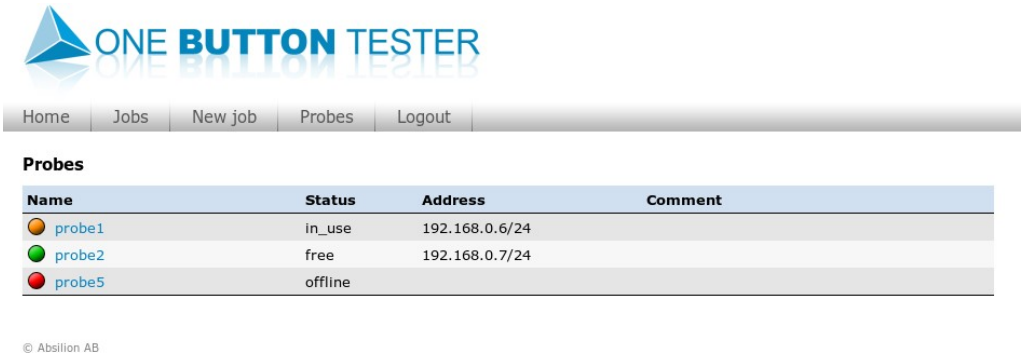

Figure 4: Probe information page for the OBT GUI

4.2.4 Create job page

The user can create new jobs as seen in Figure 5. There are several packages of tests that the user can choose from to create a job. In Figure 5 a user creates a *Monitoring BE* test and a *Monitoring BE and VoIP* test from the package *Long term monitoring*. Finally the user clicks the create button at the bottom of the page which will direct the user to the next page.

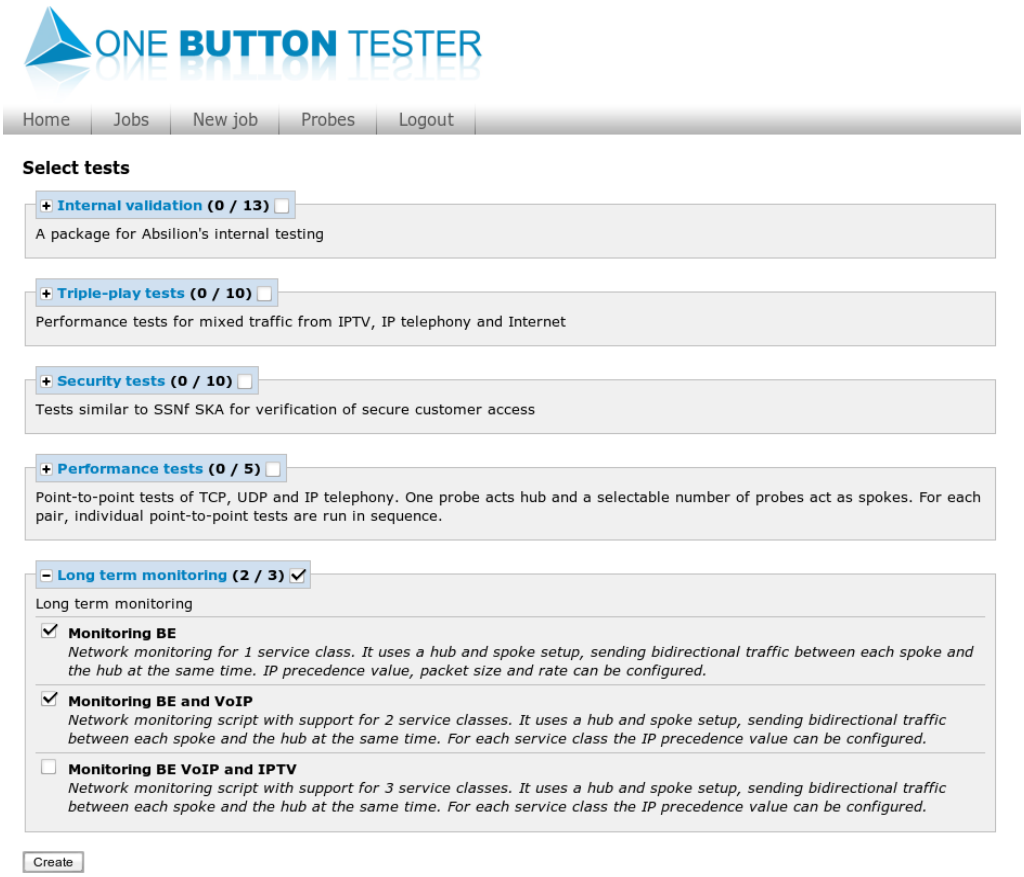

Figure 5: Create job in the OBT GUI

4.2.5 Test parameters page

The page that shows up after the user has clicked on *create* in the previous page (Figure 5) is the point at which various parameters for a test can be set. The user defines a name and a description for the test. By entering a date and time in the *scheduling* section the user can specify when to start a test. By leaving the *scheduling* section fields empty the test will execute immediately. In the *Input Variables* section the user defines which probes are to act as clients and one probe is defined to act as the hub (think *hub and spoke [19]*). In the test shown in Figure 7 a *Monitoring BE* test is performed which provides the user with an opportunity to choose IP precedence, Bitrate, Frame size and Test duration values to use for the test.

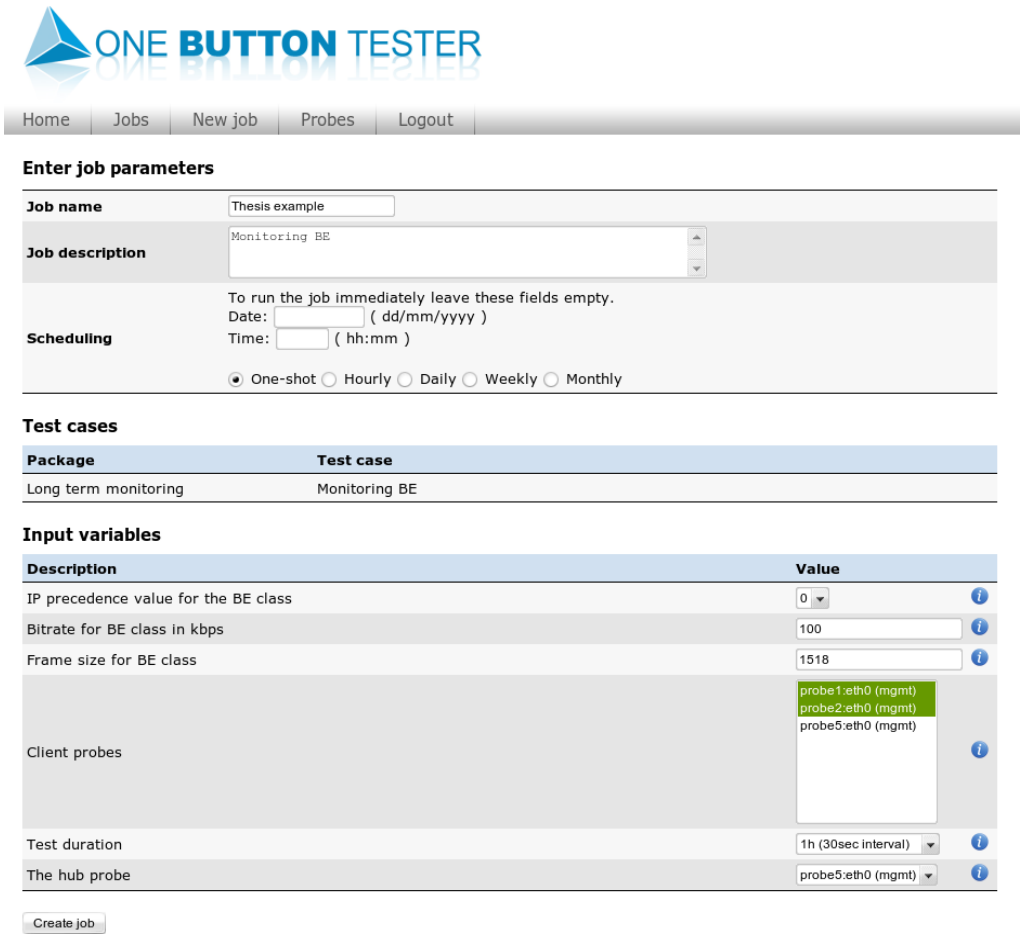

Figure 6: User defined parameters in the OBT GUI

4.2.6 Run job page

By pushing the create job button the job will start. The user can follow the test live as it runs as shown in Figure ~\ref{runjob-result}.

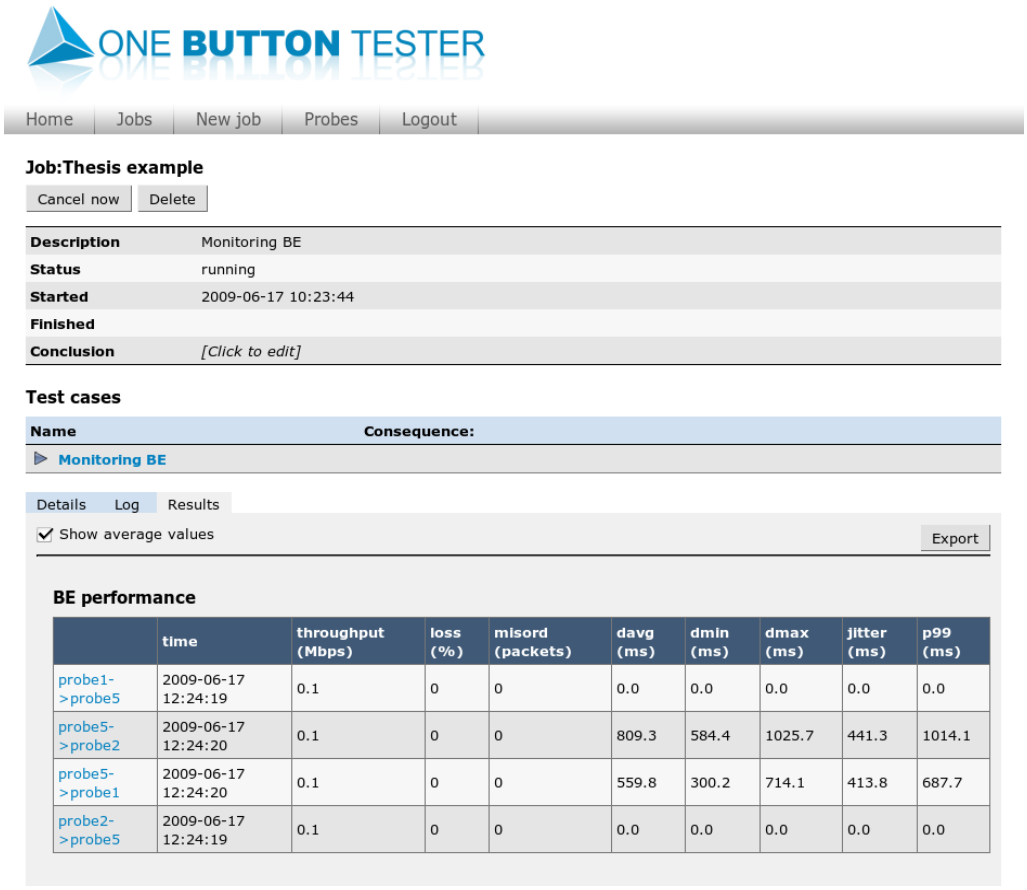

Figure 7: Value format view of test data

There is also a graphical view to view the test data in dynamically updated graphs as shown in Figure 8.

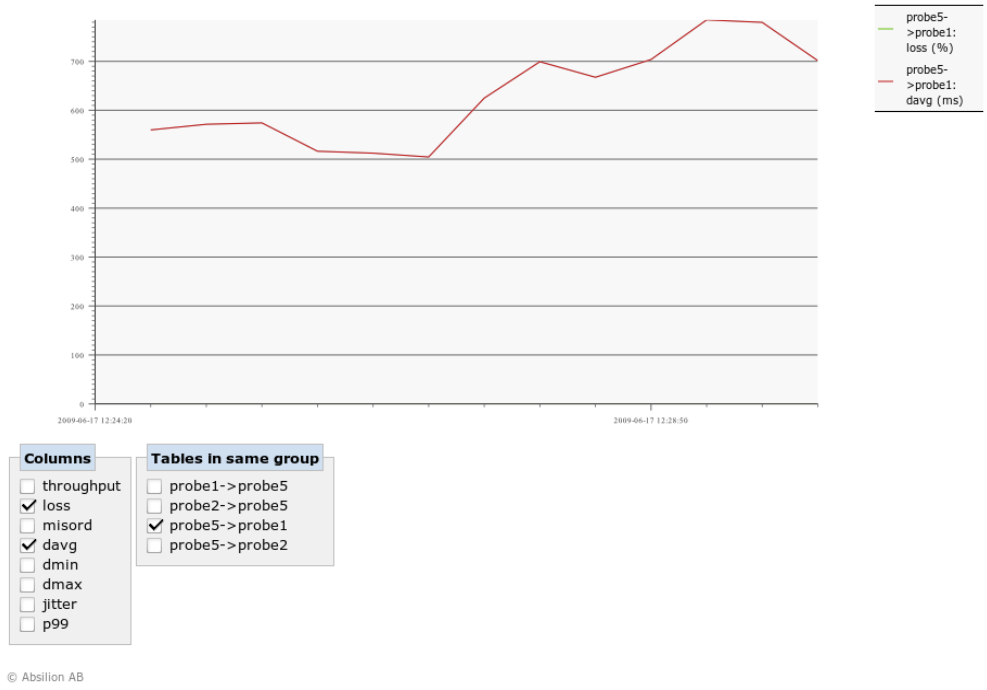

Figure 8: It is possible to have a graphical view of the test data

After the job has been completed the user will be presented with the final result. In the case of the *Monitoring BE* test this occurs if the throughput is sufficiently high.

4.3 Test Execution Control Server (TECS)

TECS is the centre of the system and is responsible for managing jobs, hosting the web server and database and keeping records of which genalyzers are connected to the system for a specific user [4]. A system overview of the TECS is shown in Figure 9.

The key feature of the whole system is the ability to have a shared TECS for a number of users (customers). In this way a user of the system merely requires two genalyzers to be installed in their network and an account within the TECS in order to use the service. No server hardware, installation, configuration, backup, update or security maintenance is required by the user~\cite{person:mats}.

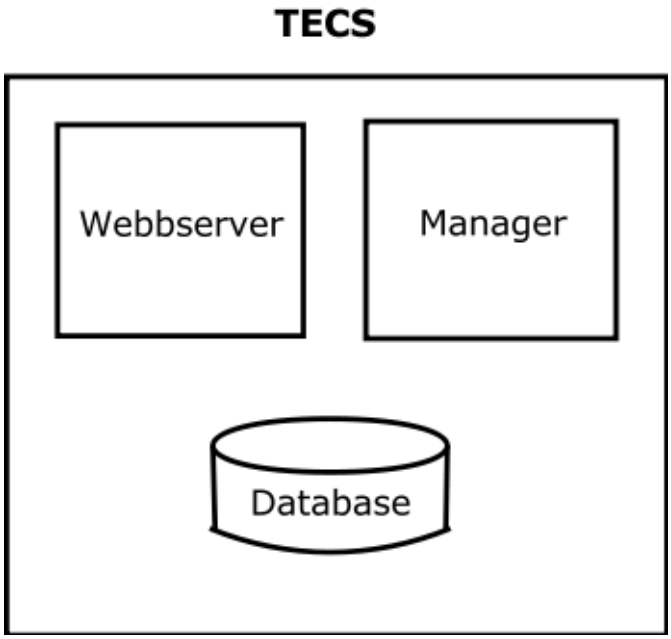

Figure 9: TECS system overview

4.3.1 Web server

The web server processes all input from the web GUI and serves the user with the necessary data retrieved from the database. The information given when a user logs onto the system is used by the web application in order to determine which settings are to be used. Various users can have different access to test packages and other features. When a logged in user creates a test job a RPyC (Remote Python Call) is established with the manager (described below) [22]. The username and a job ID are passed to the RpyC to act as arguments.

4.3.2 Manager

When a remote python call is established from the web server the manager uses the username and job ID to obtain the test scripts chosen by the user for use in that specific job. Furthermore the manager creates some objects from a module called resultobjects that is used to keep track of database storage and logging. The manager then calls the genalyzers to start sending and receiving traffic based on the configuration of the test scripts.

4.3.3 Database

The database used in OBT is a regular relational database from MySQL. The database is used to store test data, job information, test scripts and customer information. To be able to treat all data in the OBT system as objects (Object Oriented design) the Django ORM is used. This enables the developer to create ordinary Python classes for items to be stored in the database. The relevant part of the database model is shown in Figure 10 (the full database model is found in Appendix A).

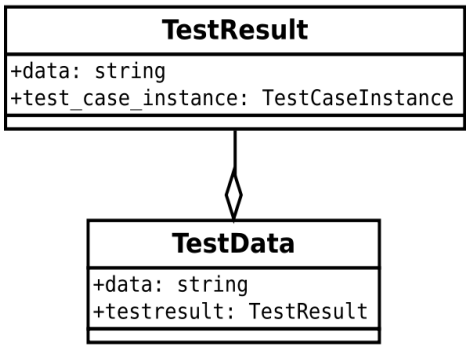

Figure 10: Extraction from database model

4.4 Genalyzer

A *genalyzer* which is an merging of the two words generator and analyzer is the physical machine placed in the network in order to test [19]. They are usually small embedded computers but the genalyzer software could also be installed on a ordinary PC but however this is conducted it is executed on a realtime kernel [20].

In a typical case the genalyzers receives an IP address via DHCP and automatically connects to the TECS. If any new updates exist for the genalyzers they will update themselves with updates provided by the TECS. Then the genalyzers are ready to be used in all the tests the user (customer) has access to [19].

4.4.1 Ntools

With Ntools a genalyzer can generate various traffic and send it over a network. Another genalyzer can act as receiver and with the use of Ntools it is possible to analyze the received traffic. In this case it becomes possible to have many genalyzers acting as both sender and receiver simultaneously with multiple streams [24].

4.5 Test scripts

The test scripts is what unites the TECS with the Genalyzers. They are defined by the maintainer of the TECS and are regular Python scripts. A simplified extract of a test script follows below.

Each script consists of a number of input variables specifying what a user of the system can set in the GUI. A example of the input variables is shown below. In the GUI this provides the user with the ability to select items such as frame size and bitrate etc. In some input variables the default values are defined but can of course be changed by the user in the GUI. In Figure 6 the GUI can be seen for the Best Effort input variables.

```
@variable: be_size = InputValue(
                  "Frame size for BE class",
                  int,
                  default=1518,
                  min=64,
                  max=1518
                  )
```

```
@variable: be_rate = InputValue(
                  "Bitrate for BE class in kbps",
                  int,
                  default=100,
                  min=100,
                  max=1000
                  )

@variable: be_prec = InputValue(
                  "IP precedence value for the BE
class",
                  int,
                  values=[0,1,2,3,4,5,6,7]
                  )
```

The test scripts are then used to set up both generator and receiver streams for the genalyzers by the use of Ntools. For each stream the test script defines objects from a class called *Table*. This object is used to keep track of the different streams and their test data. It is also responsible for the process to save test data in the database.

```
tb = objects.Table() #create Table object
```

The script also defines objects from \emph{Stream} classes which actually are ntools wrappers. One generator object that is responsible for generating network traffic is created. To receive traffic a receiver object is created.

```
gs = objects.GStream() # Generator stream
rs = objects.RStream() # Receiver stream
```

Table objects and stream objects are added to each probe forming part of the test (here defined as hub and client).

```
client.ntools.nrecv.add( rs, tb )
hub.ntools.ngen.add( gs )
```

Finally the streams are started and traffic starts to flow between the probes in the computer network.

```
hub.ntools.ngen.start()
client.ntools.nrecv.start( rs, tb )
```

5 Design

5.1 Back-end requirements specification

The back-end is the part of the system that acts before the test data is saved to the database. This is where the work of manager is carried out.

To integrate the new aggregation code into the existing system it is important to find the correct place for this to occur with respect to these criteria,

- Smart integration to current system.
- Test script management.
- No need to change old scripts to fit the new aggregation functionality.
- Find a way to save aggregated data.
- Keep database traffic as low as possible.
- Use similar coding style and solutions to common problems.

5.1.1 Smart integration into current system

As the existing system is quite complex a means of modifying it in as simple a manner as possible was sought.

Smart integration into the current system means that there should preferably be only one place at which the aggregation functionality is integrated into the current system. If modifications are made to the aggregation functionality then it should not be necessary to make any changes to the current system. This is the concept of *loose coupling* [25].

5.1.2 Test script management

As the test scripts for aggregation have to be rewritten or modified the examples regarding this must be easy to follow.

5.1.3 No need to change old scripts to fit the new aggregation functionality

If a test script has aggregation capabilities then there must be a specified means of making this decision and in such a case the aggregation code will trigger. On the other hand, if a test script does not possess any aggregation capabilities everything should work as was the case before the aggregation functionality existed. In this manner the aggregation can be integrated script by script as it occurs for particular test scripts.

5.1.4 Find a way to save aggregated data

Aggregated results data must be saved in the database because old tests should be viewable at any given time.

5.1.5 Keep database traffic as low as possible

An attempt should be made to avoid having database traffic from probes to manager. The probes have to send database traffic to the manager at least once. There should be a means of keeping it that way by forwarding the database traffic to the aggregation functionality from within the manager.

5.1.6 Use similar coding style and solutions to common problems

The new aggregation code should use a similar coding style to that of the existing system code. In some cases a problem may arise that has already been solved on the existing system. In that case the aggregation functionality should attempt to use that solution as much as possible.

If this is enforced the aggregation code will be easier for other developers to comprehend and common problems between aggregation and existing system can be solved by introducing new, loosely coupled, functionality for both modules.

5.2 Front-end requirements specification

Basically, the front-end is the GUI. But before the GUI can be presented to the end user the database has to be queried. Both the unprocessed test data and the aggregated values will be required. When all the necessary values are available to the GUI one of the thesis work main objectives will be ready to be carried out. This will involve presenting the aggregated values in such a way that it easy is to see where and when a test is failing.

- Retrieve test data and aggregation data from the database.
- Visualization that immediately shows errors in the test.
- Create aggregation levels.

5.2.1 Retrieve from database

How and when the data will be retrieved from the database. The need to keep the database traffic low is not as crucial as it was for the back-end system but the question should be considered in order to provide a better performance.

5.2.2 Visualization that immediately shows errors in the test

A user of the OBT system should be able to instantly decide upon looking at the GUI whether the test has any values that are not satisfactory.

5.2.3 Create aggregation levels

When an invalid test value shows up in the GUI the user should be able to navigate down through the aggregation abstractions (levels) back to the single test value that makes the test fail. If more than one abstraction exists (more than one level of aggregation) the user should then be able to see all the abstraction layers if this is desired by the user. However, only the top aggregation level is shown as the default as it is not usually necessary to view other levels unless the test fails.

5.3 Test script algorithms

In order to make aggregations of the unprocessed test data, algorithms have to be developed for each type of service being tested. What makes a test for Internet television is for example not the same as for Internet telephony.

5.4 Documentation

The source code developed by the author should follow the Absilion praxis for documentation and comments. In addition to the source code documentation and comments a separate documentation, for example a README file, regarding how the aggregation functionality works and how it is intended to be implemented into an existing system is required. This is to enable the aggregation functionality to be easily implemented by other developers at Absilion.

5.5 Prototypes

A backbone prototype that is constructed in the virtual test environment will be used to test different solutions with regards to how to integrate aggregation functionality into the OBT system.

When that works and the aggregated data is stored in the database a prototype regarding how to visualize the data will be designed.

5.6 Implementation

5.6.1 How to get the test data from the existing system

As the probes generate and send network traffic tests, data will be produced and this will end up in the database. Three different approaches to obtaining the test data for the aggregation functionality are evaluated.

1. **The probes send test data to aggregation**
   The first approach is to allow the probes to send test data to the aggregation. This could be performed in a similar way to that for the communication between the probes and the manager. Thus, simultaneously to the probes sending test data to the manager, the aggregation could receive the same test data in the same way. Figure 11 illustrates the path for the test data.

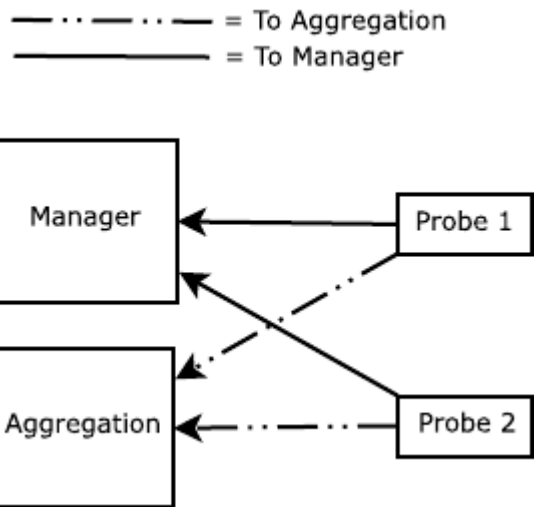

Figure 11: The probes sends test data to aggregation

2. **The manager sends test data to aggregation**
   Another way is to allow the probes and the manager to work in exactly the same manner as previously and obtain the test data on the manager side instead. Figure 12 shows the path where test data flows from the probes to the manager. The manager will process the test data for its own purposes but also send the un-processed test data to the aggregation.

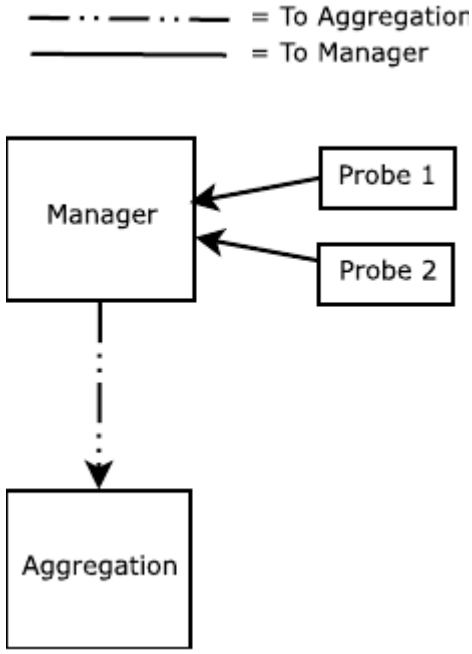

Figure 12: The manager sends test data to aggregation

3. **Get test data from the database**
   It is also the case that the test data could be retrieved directly from the database. This would require some sort of functionality that listens to the database and triggers the aggregation functionality when it is time for an aggregation calculation. This is shown in Figure 13 where the aggregation obtains the test data from the database.

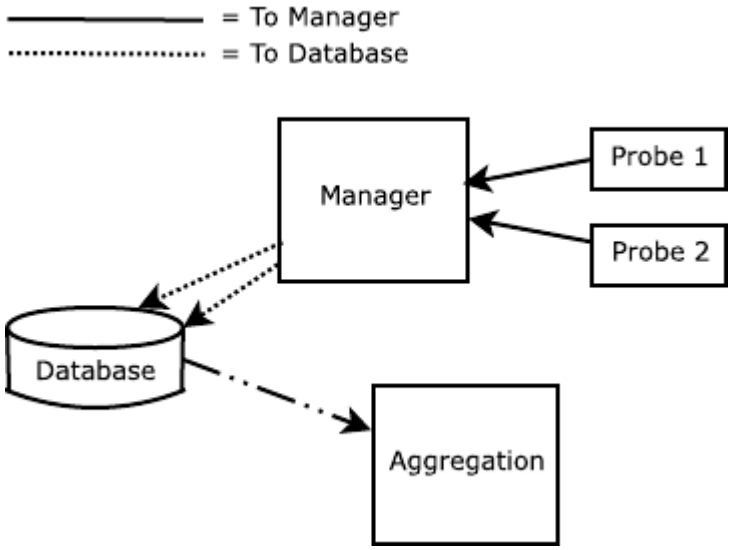

Figure 13: Get test data from the database

5.6.2 Test scripts and aggregation algorithms

The aggregation algorithms for the services in each test script will be implemented directly in the scripts. Thus it is possible for there to be extra source code lines for up to three algorithms in the test scripts used in the thesis work. The fact that new objects and comments for each algorithm are necessary should be taken into consideration as this would further increase the number of source code lines in the script.

5.6.3 How to save aggregated data

After the execution of an aggregation on a set of test data a means of storing the data is also required. This is generally conducted using either a database or in files [26]. The former is chosen as the existing system already uses a database to store test data. Three cases are evaluated with regards to how and where to store aggregation data in the database.

1. **Create new database to store aggregation data**
   A completely new database for aggregation data could be constructed. This introduces new parameters including usernames, passwords and hostnames that must be configured in the back-end to connect to this new database.

2. **Use existing database with new tables for aggregation**
   By using the existing database configuration issues such as passwords could be eliminated. A new table structure could be created which has been optimized for aggregation data.

3. **Use existing database**
   The existing database could be evaluated if it could effectively store aggregation data in the same table structure as the unprocessed test data.

5.6.4 Visualization

To construct a visualization that causes erroneous values to stand out and be easily and rapidly noticed is a non-trivial task. If it is a long test with many probes, the aggregation should take this into consideration as this will produce many aggregated values. Is the computer screen sufficiently large to show all values in a readily comprehensible way? Many values means that it is difficult to make the errors stand out and be easy to spot.

Another consideration is what will be the best way to mark an error. Is it color, shape, position, text or some combination of these that makes it easiest? Two different ways were considered in this regard.

The first is shown in Figure 14 and involves a simple bar with squares that can have different colors. If a square is green it represents an aggregation of a set of test data without any errors in the tested network. If the square is red a problem exists in the network.

Figure 14: Bar visualization

The second visualization is a diagram where both the feature that the square is green and above the line indicates that the aggregation value is satisfactory and that the network is OK for that set of test data. On the other side, if a failure in the network occurs the square is both below the line and turned red.

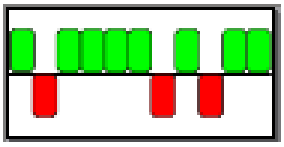

Figure 15: Color and position visualization

If there are errors in the test and a red square is present how will the user be able to find the source of the error? A solution to this is to make the red square clickable which takes the user to some lower level of aggregation or back to the original GUI developed by Absilion. It is critical to keep this visualization very simple and attempt to avoid feature creep [45].

6 Results

6.1 Retrieval of the test data to aggregate

Three different approaches regarding how to obtain test data to the aggregation functionality were proposed in the design chapter,

1. The probes send test data to the aggregation
2. The manager sends test data to the aggregation
3. Obtain test data from the database

The first option to send test data directly from the probes to the aggregation functionality has the positive effect that the aggregation functionality would be very loosely coupled to the existing system. On the other hand this functionality already exists and by implementing retrieval of test data in this manner would give rise to duplication. There would be duplication of network traffic, source code and of functionality if this approach was chosen.

The second option that was evaluated is that the manager provides the test data to the aggregation functionality. This is good because it would not increase the network traffic between the probes and the manager. This approach also enables the possibility for the reuse of existing code which is a good option. The negative part is that it increases the coupling to the existing system.

The last option was to obtain test data directly from the database. This approach would violate re-usability and would also increase the database connections which is not desirable when considering the performance of the system.

The second of these three options was chosen for the delivery of a set of test data to the aggregation functionality. This proved to be the best trade-off between re-usability and performance.

6.2 Adapt test scripts for aggregation

There are two aspects to consider in test scripts that will be adapted for the aggregation functionality.

6.2.1 Aggregation objects

The first addition to the test scripts is the introduction of objects from a new aggregation class. A model of this class is shown in Figure 16. This Aggregation class, in a similar manner to the Table class, is found in the manager.

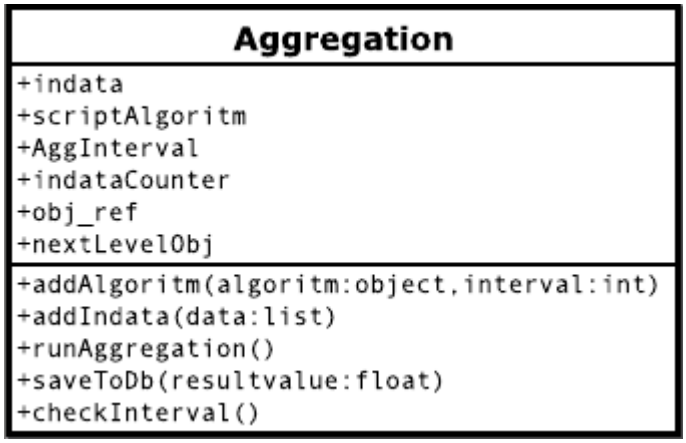

Figure 16: Aggregation class UML model

Every probe has one or more streams and for each stream there exists one Table object. Each Table object also knows of an aggregation object. This is conducted by setting a Table class member attribute as shown in Figure 17.

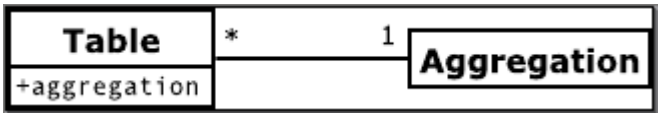

Figure 17: Table and Aggregation relationship

It is now possible for the table object to store the test data in the database and then use the aggregate object to send the same test data to the aggregate object. Note that a table object does not always have the ag-

gregate class member attribute set. If it is not set then no aggregation should be conducted for this type of test data (the test does not have aggregation functionality).

6.2.2 Aggregation algorithms

The other change to the test scripts is the need to define the aggregation algorithms.

As everything in Python is objects, including functions, the solution was to define the required aggregation algorithms as functions in the test script [28]. Then those functions (algorithms) were sent as objects to the aggregation objects and set as member attributes (see *scriptALgoritm* data attribute in Figure 16).

An algorithm that aggregates test data for Internet telephony might have the following appearance,

```
def voipAlgoritm(rows):
                  mosSum = 0
                  rowcounter = 0
                  for r in rows:
                   if r[9] < 4:
                     return false
                   mos = r[9]
                   mosSum += mos
                   rowcounter += 1

                  if rowcounter != 0:
                   res = round((mosSum/rowcounter),2)
                   return res
                  else:
                   return 'Error'
```

And to add this algorithm to the aggregate member attribute,

```
# Create aggregate object
ag = objects.Aggregate()

# Set member attribute with algorithm
ag.addAggrAlgoritm( voipAlgoritm )
```

6.3 Execute the aggregation

The aggregation class has a member attribute called *AggInterval* as seen in Figure 16. By setting this member attribute the aggregation functionality knows how many unprocessed test data values should be aggregated. One way to set this interval is to send it as a parameter to the *addAggrAlgoritm* function described above. In the example below the *AggInterval* 4 is set. This specifies that 4 unprocessed test data should be received before an aggregation occurs.

```
# Set member attribute with algorithm and interval
ag.addAggrAlgoritm( voipAlgoritm, 4 )
```

6.4 Database storage for aggregated values

As the decision was made to reuse the existing system components to obtain the test data from the probes it is natural to reuse the manner in which data is saved in the database. The unprocessed test data is saved in a serialized form by the Python module *Pickle* [29]. This could easily be used to save aggregated data in the same way. The database does not care if it is a serialized unprocessed test data or an aggregation value that is stored.

The relevant part of the database design for storing aggregation values is the TestData and TestResult tables shown in Figure 18.

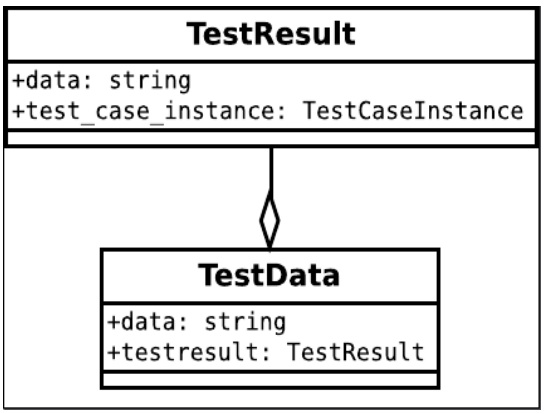

Figure 18: Table and Aggregation relationship

The *TestResult* data attribute is actually the pickled aggregation object. By means of an foreign key relationship the aggregation object maps to a number of values saved in the *TestData* table [30]. This is a similar approach as that already used by the table objects for the database storage.

6.5 From database to GUI

To obtain the aggregated values from the database and present them to the user some modifications were required to the existing system. The view in the existing system looks in the database for pickled objects from the table class to be presented to the user. An extension in the view to also look for aggregation objects in the database was developed.

If the view finds aggregated values it sends those values to the template. The templates had to be modified to present aggregated values if such existed. But once again the system should behave as was the case before the aggregation functionality was available if a particular test does not support aggregation. Therefore some more logic had to be implemented in the templates to display different aspects depending on whether or not the aggregation functionality was present.

6.6 Visualization of aggregated test data

From the two alternatives presented in the design chapter the simple bar approach was chosen as shown in Figure 14. The reason was simply that it was the first visualization that was tested and it satisfied the requirements to be easy to comprehend and to enable the end user to rapidly spot errors in the test. During the implementation it was also discovered that it would be relatively easy to modify this simple bar to more sophisticated visualizations if required.

When implemented in the virtual test environment the result is as shown in Figure 19. The green squares indicate that the tested network is satisfactory for that set of test data that the aggregation has been applied to. The red squares indicate problems with one or more values in the set of test data that the aggregation represents. This should be investigated closer by the end user to determine exactly what the problem is.

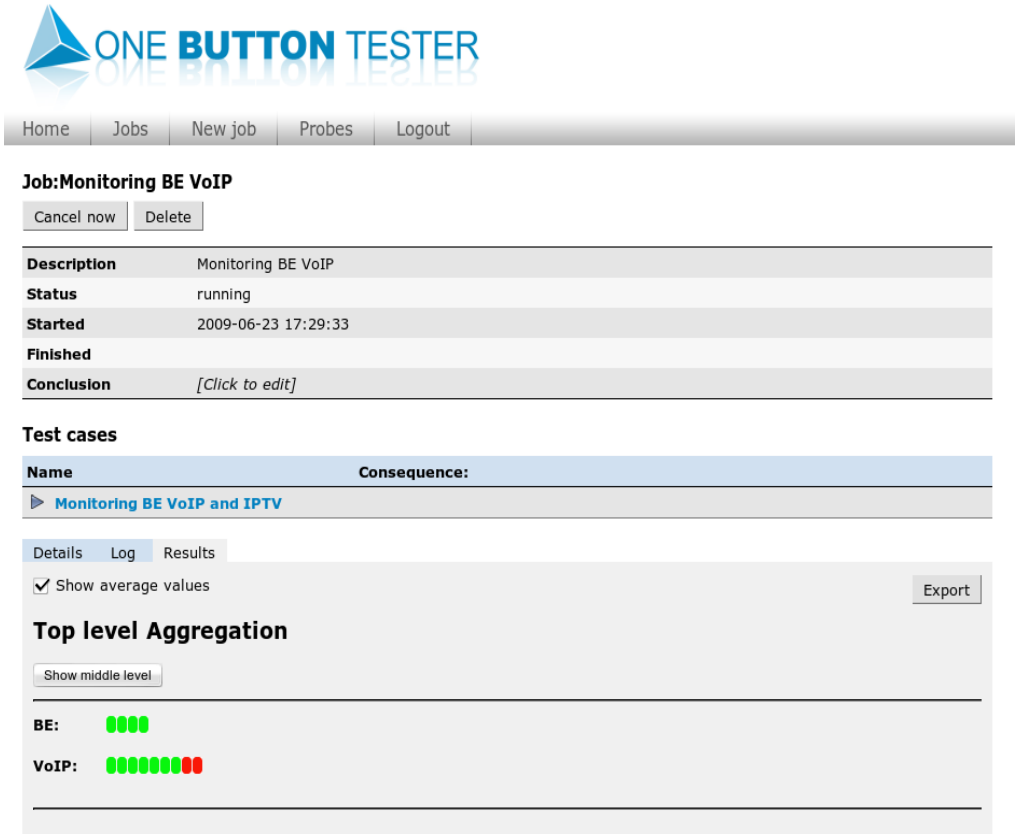

Figure 19: Implemented bar visualization

6.7 Levels of aggregation

As mentioned above the end user should investigate what the problem is when a red square is present in the bar visualization. This is performed by levels of aggregations. Figure 19 shows the top level of aggregation.

In this case the top level is one of three levels and is an aggregation of all the probes and their streams for a best effort test and Internet telephony test. By clicking on the button Show middle level (or any red square if there are any) in the GUI the user will be taken to the level below (middle level).

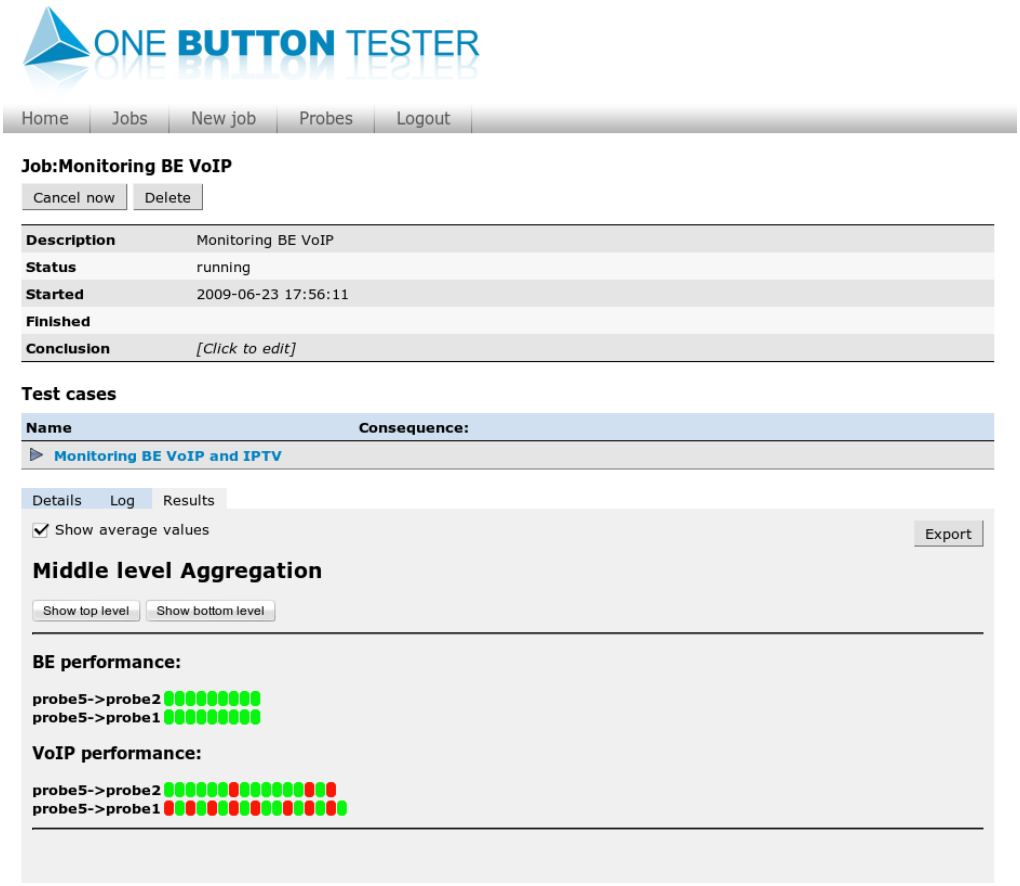

Figure 20: Implemented bar visualization

In Figure 20 the middle level shows an aggregation of all the test data between different probes. For example the BE performance now shows two bar visualizations. One bar for the test between probe 5 and probe 1 and one bar for test between probe 5 and probe 2.

7 Conclusions

7.1 Results discussion

7.1.1 Obtain an understanding of the existing OBT system architecture and usage

By making relatively small changes in the existing OBT system a working aggregation functionality could be implemented as proved in the results chapter. To achieve this an understanding of the OBT was required.

7.1.2 Collect test data from the OBT system to use in aggregation

The solution to this problem proved to be simpler than at first appeared. As the OBT system was so easy to understand this meant that is was possible to enter into the source code to make some minor changes. This made it possible to modify the code relatively easily to fit the aggregation functionality with only a minor risk of breaking some other functionality. In other words, it was possible to reuse a great deal of existing code to collect the test data to be aggregated.

7.1.3 Modify existing testscripts to support aggregation functionality

The concept of using aggregation objects to keep track of probes and streams proved to be a successful approach. This made it possible to keep testscripts without aggregation intact and add aggregation functionality whenever it was required. This was performed without having to adjust all the testscripts because one test required aggregation.

7.1.4 Find a way to calculate aggregation of testdata depending on which type of test that's performed

As the aggregation algorithms involved the use of Python functions any algorithm could be used for whatever testscript are required. Thus by checking the type of test a specific aggregation could be selected from all the algorithms. This also makes it easy to reause algorithms in different testscripts.

7.1.5 Make it possible to store aggregation results in a database

Once again an existing solution could be reused. The developers at Absilion had made a good decision by saving the test data as serialized objects in the database. Since the aggregation functionality makes use of aggregation objects the results from the aggregation could be stored in the database in a similar manner to that for ordinary test data.

7.1.6 Retrieve aggregated data from database and make it available in the web page dynamically

Since the decision has been made to reuse existing code and functions, it was natural to continue using this method with regards to the retrieval of aggregated data from the database.

7.1.7 Construct an easy to understand visualization of the aggregation result

The visual result, the bar visualization, might not be the most elegant way to represent how the test is progressing. However, it is important to note that it enables the job to be performed in a satisfactory manner. It is fairly certain that the end user is not particularly concerned if shadow or other effects are present. It is really the case that all that is required is to determine as rapidly as possible whether or not the test is satisfactory.

7.1.8 Create levels of aggregation

In the prototype presented in this report two levels of aggregation have been tested. By using the aggregation class as an base class almost any number of levels of aggregation could be created easily through class inheritance.

7.1.9 Make a easy to use navigation between levels of aggregation and test data on web page

It is always difficult to determine just how easy it is to use something. Typically this requires real users to conduct tests on the system as it is not easy to second guess how users who are unfamiliar with the inner workings of the system and web page will react.

7.2 Overall thoughts

The project has proven to be very interesting and instructive both in the field of computer networks and to the main purpose of system development and programming. New experiences have also been gained in more practical subjects such as version control (SVN), virtualization (VirtualBox) and project management systems (Trac).

It has been surprisingly easy to build an understanding of the existing OBT system partly because it is a well written system and partly because the Python programming language is used. Also the other developers at Absilion have been very helpful and willing to provide explanations.

A great deal of existing code has been reused from the OBT system in this thesis work which has been a positive asset. Re-usability is good (in most cases), and code that could be reused is often good code.

7.3 Future work

The most notable aspect regarding further development is the test scripts and in particular in relation to handling different aggregation algorithms, how to reuse more code and have less duplication within the test scripts. It might be possible to create more of an object oriented approach for test scripts and to make use of class inheritance and polymorphism. Another thesis work could be to make an GUI based test-script generator.

Another interesting aspect would be to perform real user tests where actual users of the system sit down and try out the system and the GUI. This is by far the best way to determine whether or not the aggregation functionality and its GUI are good or bad.

References

[1] B. A. Forouzan, *Data communications and Networking*, p. 775. McGraw-Hill, fourth ed., 2007.

[2] "Triple play (telecommunications)." http://en.wikipedia.org/wiki/Triple_Play_(telecommunications). Retrieved 2009-10-18.

[3] "Quality of Service" http://en.wikipedia.org/wiki/Quality_of_service. Retrieved 2009-10-26

[4] "Test sysem as a service" http://www.absilion.com/en/test-system-as-a-service.html" Retrieved 2009-10-26.

[5] "Django homepage." http://djangoproject.com Retrieved 2010-04-21.

[6] "Python homepage." http://www.python.org Retrieved 2010-04-21.

[7] "Virtualbox documentation." http://www.virtualbox.org/manual/UserManual.html#networkingdetails. Retrieved 2009-11-08.

[8] "Virtualbox documentation." http://www.virtualbox.org/manual/UserManual.html#sharedfolders. Retrieved 2009-11-09.

[9] "Applications for python." http://www.python.org/about/apps/. Retrieved 2009-11-09.

[10] "Wikipedia python article." http://en.wikipedia.org/wiki/Python_(programming_language) Retrieved 2009-11-09.

[11] "Djangobook 2.0." http://www.djangobook.com/en/2.0/chapter-05/. Retrieved 2009-11-09.

[12] "Djangoproject overview webpage." http://docs.djangoproject.-com/en/dev/intro/overview/. Retrieved 2009-11-09.

[13] "Djangobook 2.0." http://www.djangobook.com/en/2.0/chapter-04/. Retrieved 2009-11-09.

[14] "Bredband." http://sv.wikipedia.org/wiki/Bredband. Retrieved 2009-11-10.

[15] "Supported tests." http://www.absilion.com/en/test-system-as-a-service/supported-tests.html Retrieved 2009-11-10.

[16] "Best effort." http://en.wikipedia.org/wiki/Best_effort_delivery. Retrieved 2009-11-10.

[17] "Internet telephony." http://en.wikipedia.org/wiki/IP_telephony. Retrieved 2009-11-10.'

[18] "Mean opionion score." http://en.wikipedia.org/wiki/Mean_Opi-onion_Score. Retrieved 2009-11-10.

[19] M. Nordlund, Internal company meetings at Absilion AB held in 2009-03-18, Luleå.

[20] "IPTV." http://en.wikipedia.org/wiki/iptv. Retrieved 2009-11-21

[21] "Absilion instruction video." http://www.absilion.com/se/test-system-as-a-service/videoklipp.html.
Retrieved 2009-11-21

[22] "Remote python call." http://rpyc.wikidot.com/start.
Retrieved 2009-11-29

[23] "Netrounds in brief." http://www.absilion.com/en/test-system-as-a-service/netrounds-in-brief.html.
Retrieved 2009-11-23

[24] N. Vegh, "Ntools." http://norvegh.com/ntools/index.php
Retrieved 2009-11-23

[25] "Loose coupling and cohesion." http://c2.com/cgi/wiki?CouplingAndCohesion
Retrieved 2009-11-26

[26] T.R. Thomas Padron-McCarthy, *Databasteknik*, p. 9. Lund: Studentlitteratur, 2005

[27] "Feature creep." http://en.wikipedia.org/wiki/Feature_creep
Retrieved 2009-11-28

[28] "Everything is an object." http://www.diveintopython.org/getting_to_know_python/everything_is_an_object.
Retreived 2009-11-29

[29] "Pickle module." http://docs.python.org/library/pickle.html
Retreived 2009-11-28

[30] "Foreign key." http://en.wikipedia.org/wiki/Foreign_key
Retreived 2009-11-29

Appendix A: Database model